Problem Set 3 **yl2nr**

DATA STRUCTURES AND ALGORITHMS 2 - FALL 2023 DUE OCTOBER 20, 2023 AT 11:59 PM

Collaborators: list your collaborators

Sources: list your sources

PROBLEM 1 *Scenic Highways*

In this problem we'll describe something similar to solving a problem with Dijkstra's algorithm, and ask you some questions about this new problem and a possible algorithm.

You are taking a driving vacation from town to town until you reach your destination, and you don't care what route you take as long as you maximize the number of scenic highways between towns that are part of your route. You model this as a weighted graph G, where towns are nodes and there is an edge for each route between two towns. An edge has a weight of 1 if the route is a scenic highway and a 0 if it is not.

You need to find the path between two nodes s and t that maximizes the number of scenic highways included in the path. You design a greedy algorithm that builds a tree like Dijkstra's algorithm does, where one node is added to a tree at each step. From the nodes adjacent to the tree-nodes that have already been selected, your algorithm uses this greedy choice:

Choose the node that includes the most scenic highways on the path back to the start node s.

When node t is added to the tree, stop and report that path found from s to t. Reminder (in case it helps you): the greedy choice for Dijkstra's algorithm was:

Choose the node that has the shortest path back to the start node s.

1. What is the key-value stored for each item in the priority-queue? Also, what priority-queue operation must be used when we need to update a key for an item already stored in the priority-queue?

   Solution:

   1. key-value pair should be (town (vertex), weight of route (edge)). All the key-value pairs should have $-\infty$ as the value to initiate.

   2. The operation should be IncreaseKey(), so that we can update the value of the key and get a new sorted queue with the max at the first. We also need getMax() operation to get the route with the scenic highway.

2. Draw one or more graphs to convince yourself that the greedy approach describes above does not work. Then explain in words as best you can the reason this approach fails or a situation that will cause it to fail (i.e., a counterexample).

   Solution:

   This algorithm fails as when we apply the greedy selection on each substructure, the substructure may not be the optimal one. In other words, at each substructure, we cannot be aware of the options in the next substructure to choose, so it may not be the optimal substructure.

Assume we are at node a, and a has edges to b and c.

b is then connected to d, d to e, and e to t.

c is then connected to t only.

If the weight of (a,b) is 0 but the weight of (a,c) is 1, then by the greedy algorithm we will select (a,c).

However, the weights of (b,d), (d,e), and (e,t) are all 1, but the weight of (c,t) is 0. So we know that we should choose (a,b) instead of (a,c) as b would lead to more scenery highways (higher total weights) of 3, but (a,c) only leads to 1 before we end up at t.

This example shows that the algorithm may not give the optimal solution all the time.

PROBLEM 2 *As You Wish*

Buttercup has given Westley a set of n tasks $T = t_1, \dots, t_n$ to complete on the farm. Each task $t_i = (d_i, w_i)$ is associated with a deadline d_i and an estimated amount of time w_i needed to complete the task. To express his undying love to Buttercup, Westley strives to complete all the assigned tasks as early as possible. However, some deadlines might be a bit too demanding, so it may not be possible for him to finish a task by its deadline; some tasks may need extra time and therefore will be completed late. Your goal (inconceivable!) is to help Westley minimize the deadline overruns of any task; if he starts task t_i at time s_i, he will finish at time $f_i = s_i + w_i$. The deadline overrun (or lateness) of tasks—denoted L_i—for t_i is the value

$$L_i = \begin{cases} f_i - d_i & \text{if } f_i > d_i \\ 0 & \text{otherwise} \end{cases}$$

Design a polynomial-time algorithm that computes the optimal order W for Westley to complete Buttercup's tasks so as to minimize the maximum L_i across all tasks. That is, your algorithm should compute W that minimizes

$$\min_{W} \max_{i=1,\dots,n} L_i$$

In other words, you do not want Westley to complete *any* task *too* late, so you minimize the deadline overrun of the task completed that is most past its deadline.

1. What is your algorithm's greedy choice property?

   Solution:

   The general idea is to prioritize tasks with the earliest deadline. So at each time t_0, choose to start the tasks with the least d (earliest deadline) among all the tasks that have positive overruns ($L > 0$). If no task has a positive overrun, then start with the task with the least d.

2. What is the run-time of your algorithm?

   Solution:

   The time complexity is $O(n \log n)$. Each sorting takes $\log n$ time and there are at most n times of sorting.

3. Consider the following example list of tasks:

   $$T = \{(2,2), (11,1), (8,2), (1,5), (20,4), (4,3), (8,3)\}$$

   List the tasks in the order Westley should complete them. Then argue why there is no better result for the given tasks than the one your algorithm (and greedy choice property) found.

   Solution:

   Sequence: (1,5),(2,2),(4,3),(8,3),(8,2),(11,1),(20,4)

   We can show that this sequence is optimal by contradiction. Let's say there is another optimal sequence with a difference at two pairs: (d_j, w_j) comes before (d_k, w_k).

   As it is different from my sequence, at the start of t_j, with current time s_0,
   either 1.$s_0 + w_j - d_j <= 0$ and $s_0 + w_k - d_k > 0$ or 2.both overruns are positive but $d_j > d_k$.

   Then the new sequence would generate a maximum overrun of $L_{new} = \max((s_0 + w_j - d_j), (s_0 + w_j + w_k - d_k))$

and my original sequence generate a overrun of $L_{original} = \max((s_0 + w_k - d_k), (s_0 + w_k + w_j - d_j))$ when we finish the latter task

if it's case 1, then $L_{new} = (s_0 + w_j + w_k - d_k) > L_{original} = (s_0 + w_k - d_k)$.

If it's case 2, then $L_{new} = (s_0 + w_j + w_k - d_k) > L_{original} = (s_0 + w_k + w_j - d_j)$ as $d_k < d_j$.

Thus, the new sequence has a larger maximum overrun than the original sequence, so there is no better sequence than the original one. The original sequence is the optimal one.

PROBLEM 3 *Course Scheduling*

The university registrar needs your help in assigning classrooms to courses for the spring semester. You are given a list of n courses, and for each course $1 \leq i \leq n$, you have its start time s_i and end time e_i. Give an $O(n \log n)$ algorithm that finds an assignment of courses to classrooms which minimizes the *total number* of classrooms required. Each classroom can be used for at most one course at any given time. Prove both the correctness and running time of your algorithm.

Solution:

1. We sort the list of courses by the ascending order of end time. For those with the same end time, we then sort according to the ascending order of start time.

We then create a list of classrooms with the latest end time for each classroom. The list is initially empty

Pop out the first course with s_1, e_i in the sorted list of courses, we check if there is a classroom in the list available for the course (classroom with the latest end time before the start time s_1). If there is an available classroom in the list, we add the course to that classroom by updating the latest end time of that classroom to be e_1. If there is no classroom available, we then add a new classroom to the list and update its latest end time accordingly.

2. To prove this algorithm is correct, we first show that there are no overlapping courses. By sorting the list of courses by their end times, we ensure the later courses we retrieve from the list must have later end times than previous popped courses. As we update the classroom with the latest end time, the newly retrieved course must have a later end time. Thus, if the start time of the newly retrieved course is before the latest end time of the classroom, the newly retrieved course must overlap with the current course in that classroom, and we detect that conflict and avoid it by adding a new classroom. Thus, the algorithm ensures that there are no overlapping courses in the same classroom.

We then show it minimizes the number of classrooms. This follows that we detect overlapping time conflicts of courses in the same classroom. As every time we add a new classroom means that none of the existing classrooms can hold this course, there is no way to decrease the number of classrooms used. Thus, the algorithm generates the minimized number of classrooms.

To prove the running time of the algorithm, we can apply merge sort or quick sort to sort the courses, so it is $O(\log n)$. For the checking step, we can apply a binary heap to store the latest end times of classrooms. Then to check if there is an available classroom is also $O(\log n)$. Finally, there are n courses, so the total running is $O((n+1)\log n) = O(n \log n)$.

PROBLEM 4 *Ubering in Arendelle*

After all of their adventures and in order to pick up some extra cash, Kristoff has decided to moonlight as the sole Uber driver in Arendelle with the help of his trusty reindeer Sven. They usually work after the large kingdom-wide festivities at the palace and take everyone home after the final dance. Unfortunately, since a reindeer can only carry one person at a time, they must take each guest home and then return to the palace to pick up the next guest.

There are n guests at the party, guests $1, 2, ..., n$. Since it's a small kingdom, Kristoff knows the destinations of each party guest, $d_1, d_2, ..., d_n$ respectively and he knows the distance to each guest's destination. He knows that it will take t_i time to take guest i home and return for the next guest. Some guests, however, are very generous and will leave bigger tips than others; let T_i be the tip Kristoff will receive from guest i when they are safely at home. Assume that guests are willing to wait after the party for Kristoff, and that Kristoff and Sven can take guests home in any order they want. Based on the order they choose to fulfill the Uber requests, let D_i be the time they return from dropping off guest i. Devise a greedy algorithm that helps Kristoff and Sven pick an Uber schedule that minimizes the quantity:

$$\sum_{i=1}^{n} T_i \cdot D_i.$$

In other words, they want to take the large tippers the fastest, but also want to take into consideration the travel time for each guest. Prove the correctness of your algorithm. (Hint: think about a property that is true about an optimal solution.)

Solution:

As T_i is fixed for each guest, to minimize $\sum_{i=1}^{n} T_i \cdot D_i$ means to find the order that minimizes D_i for large T_i. So the idea is to find the guests who give more tips per time and serve them first to make the product of tips and time smaller.

1. We sort the list according to the descending order of the quantity $\frac{T_i}{t_i}$. Note $D_i = t_i + t_{previousguests}$, with $t_{previousguests} = 0$, so $D_i = t_i$
2. Starting with $t_{previousguests} = 0$, we pop out the first guest k from the sorted list. Append the guest k to the Uber schedule. Then we update the $t_{previousguests} = t_{previousguests} + t_k$.
3. Repeat the sorting and popping out steps until the list of guests is empty.
4. Return the final Uber schedule.

This is correct because at every substructure we select the one with the largest tip per time quantity, and by prioritizing that guest, minimizing time, we ensure every $T_i \cdot D_i$ is minimized. As every element in the summation is minimized, the final result is also minimized and optimal.

Thus, we can prove by Exchange Argument. Assume there is an optimal sequence s. Then consider at any time node t, guest with T_s is the next to be served according to s. However, if we exchange guest with T_s to the guest with T_a which is the guest selected according to our algorithm, then the change in the result is $T_a \cdot (t_{previousguests} + t_a) + T_s \cdot (t_{previousguests} + t_a + t_s) - T_s \cdot (t_{previousguests} + t_s) + T_a \cdot (t_{previousguests} + t_s + t_a) = T_s * t_a - T_a * t_s <= 0$ as we know $\frac{T_a}{t_a} >= \frac{T_s}{t_s} \implies T_a * t_s >= T_s * t_a$. So exchanging the guest to the one in our algorithm does not make the case worse, and it shows that the algorithm is correct.